

This Page Is Inserted by IFW Operations  
and is not a part of the Official Record

## **BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

**IMAGES ARE BEST AVAILABLE COPY.**

**As rescanning documents *will not* correct images,  
please do not report the images to the  
Image Problem Mailbox.**

## PATENT ABSTRACTS OF JAPAN

(11)Publication number : 06-250847

(43)Date of publication of application : 09.09.1994

(51)Int.Cl.

G06F 9/45

G06F 9/44

G06F 15/16

(21)Application number : 05-036601

(71)Applicant : FUJITSU LTD

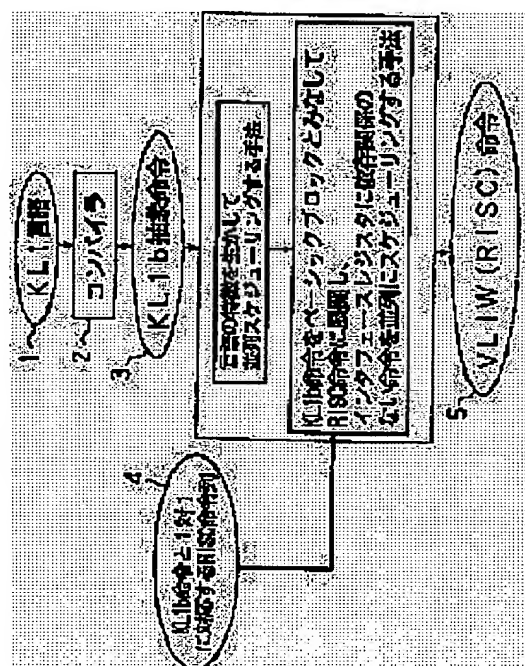
(22)Date of filing : 25.02.1993

(72)Inventor : SAKAMOTO MARIKO

**(54) CONVERSION SYSTEM FOR CONVERTING INSTRUCTION OF PARALLEL LOGIC TYPE LANGUAGE INTO VLIW TYPE INSTRUCTION****(57)Abstract:**

**PURPOSE:** o provide a conversion system for converting an instruction of parallel logic type language such as GHC and KL1 into an VLIW instruction to be executed by a VLIW type parallel computer.

**CONSTITUTION:** A parallel logic type language 1 such as a KL1 language is compiled by a compiler 2 to generate a KL1b abstract instruction 3. Then the instruction 3 is scheduled by utilizing the feature of the parallel logic type language. On the other hand, the parallel pattern 4 of an RISC instruction string corresponding to the instruction 3 at the rate of 1 to 1 is previously set up and the instruction 3 is developed to an RISC instruction string. In this case, the depending relation of an interface register is checked by regarding the instruction 3 as a basic block and the instruction 3 having no depending relation is scheduled in parallel.

**LEGAL STATUS**

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

(19)日本国特許庁 (J P)

(12) 公 開 特 許 公 報 (A)

(11)特許出願公開番号

特開平6-250847

(43)公開日 平成6年(1994)9月9日

(51)Int.Cl. <sup>5</sup>	識別記号	庁内整理番号	F I	技術表示箇所
G 0 6 F 9/45				
9/44	3 3 0 B	9193-5B		
15/16	4 3 0	9190-5L		
		9292-5B		
			G 0 6 F 9/ 44	3 2 2 F

審査請求 未請求 請求項の数4 O L (全 17 頁)

(21)出願番号 特願平5-36601

(22)出願日 平成5年(1993)2月25日

(71)出願人 000005223

富士通株式会社

神奈川県川崎市中原区上小田中1015番地

(72)発明者 坂本 真理子

神奈川県川崎市中原区上小田中1015番地

富士通株式会社内

(74)代理人 弁理士 京谷 四郎

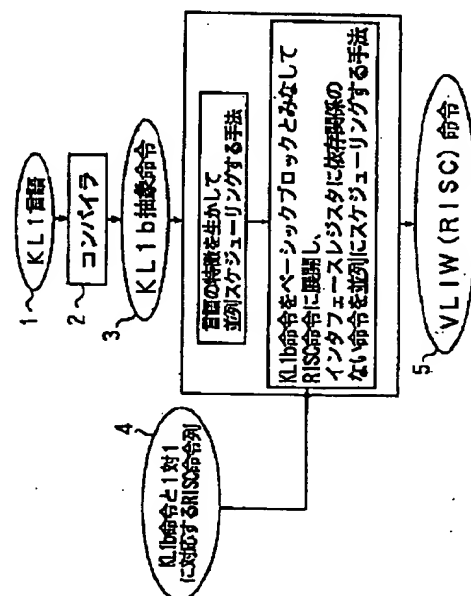
(54)【発明の名称】 並列論理型言語の命令をV L I W方式の命令に変換する変換方式

(57)【要約】

【目的】 G H C, K L 1 などの並列論理型言語の命令を、V L I W方式の並列計算機で実行できるようなV L I W命令に変換する変換方式を提供すること。

【構成】 K L 1 言語などの並列論理型言語1をコンパイラ2によりコンパイルしてK L 1 b抽象命令3を生成する。ついで、並列論理型言語の特徴を生かしてK L 1 b抽象命令3を並列にスケジューリングする。一方、K L 1 b抽象命令3に一つ一つに対応するR I S C命令列の並列パターン4を設定しておき、K L 1 b抽象命令をR I S C命令列に展開する。その際、K L 1 b抽象命令3をベシック・ブロックとみなして、インタフェース・レジスタの依存関係を調べ、依存関係のないK L 1 b抽象命令3を並列にスケジューリングする。

本発明の原理説明図



## 【特許請求の範囲】

【請求項1】 抽象度の高い並列論理型言語の命令を、命令レベルで並列実行を行うVLIW方式の並列計算機で実行できる機械語レベルに近いRISC命令列に変換する変換方式において、

並列論理型言語をコンパイルして、並列論理型言語の命令を生成し、

生成された並列論理型言語の命令を並列論理型言語の特徴を生かして並列にスケジューリングし、

スケジューリングされた並列論理型言語の命令を、並列論理型言語の命令に一对一に対応したRISC命令列に展開することにより、並列論理型言語をVLIW方式の命令に変換することを特徴とする並列論理型言語の命令をVLIW方式の命令に変換する変換方式。

【請求項2】 並列論理型言語の命令に一对一に対応させて、RISC命令を展開し、展開されたRISC命令列の並列性を解析して、並列に実行できる命令を横に並べて記述することにより、あらかじめ、並列論理型言語の命令に一对一に対応した、RISC命令列の並列パターンを設定しておくことを特徴とする請求項1の並列論理型言語の命令をVLIW方式の命令に変換する変換方式。

【請求項3】 並列論理型言語の命令をベーシック・ブロックと見なし、ベーシック・ブロックの融合の可否を並列論理型言語の仕様にある並列性とインタフェース・レジスタの依存関係に基づき判定することにより、並列論理型言語の命令を並列に記述することを特徴とする請求項1の並列論理型言語の命令をVLIW方式の命令に変換する変換方式。

【請求項4】 ベーシック・ブロック融合する際に、ベーシック・ブロックの中の、並列に実行される可能性のある並列論理型言語の命令を実行するRISC命令の依存関係を調査することなく、並列なRISC命令列を生成することを特徴とする請求項3の並列論理型言語の命令をVLIW方式の命令に変換する変換方式。

## 【発明の詳細な説明】

## 【0001】

【産業上の利用分野】本発明は、抽象度の高い並列論理型言語の命令をVLIW方式の計算機で高速に実行するため、並列論理型言語の命令をVLIW命令に変換する変換方式に関する。並列論理型言語での並列処理は、後述するように複数のゴールを同時に実行する方式で実現されている。それに対して、VLIW方式の並列処理は、ゴールより機械命令に近いレベルで命令を同時に実行して実現される。

【0002】本発明は抽象度の高い並列論理型言語の命令を、より機械語に近いレベルであるRISC命令に展開し、また、その展開の過程で並列化のスケジューリングを行う、並列論理型言語の命令をVLIW命令に変換する変換方式に関するものである。

## 【0003】

【従来の技術】従来、GHC、KL1などの並列論理型言語は単一の逐次実行プロセッサ、あるいは、逐次実行のプロセッサを複数接続したシステムで実行されていた。プロセッサを複数接続したシステムでは、いわゆる並列実行が実現されていたが、この場合の並列実行はゴール・レベルでの並列実行であり、命令レベルの並列実行は実現されていなかった。

## 【0004】

10 【発明が解決しようとする課題】本発明は、上記した従来技術を考慮して発明されたものであって、GHC、KL1などの並列論理型言語の命令を、VLIW方式の並列計算機で実行できるようなVLIW命令に変換する変換方式を提供することを目的とする。

## 【0005】

20 【課題を解決するための手段】図1は本発明の原理説明図である。上記課題を解決するため、本発明の請求項1の発明は、図1に示すように、抽象度の高い、例えば、KL1言語のKL1b抽象命令などの並列論理型言語の命令を、命令レベルで並列実行を行うVLIW方式の並列計算機で実行できる機械語レベルに近いRISC命令列に変換する変換方式において、並列論理型言語KL1をコンパイルして、並列論理型言語の命令KL1bを生成し、生成された並列論理型言語の命令KL1bを並列論理型言語の特徴を生かして並列にスケジューリングし、スケジューリングされた並列論理型言語の命令KL1bを、並列論理型言語の命令KL1bに一对一に対応したRISC命令列に展開することにより、並列論理型言語をVLIW方式の命令に変換するようにしたものである。

30 【0006】本発明の請求項2の発明は、請求項1の発明において、並列論理型言語の命令KL1bに一对一に対応させてRISC命令を展開し、展開されたRISC命令列の並列性を解析して、並列に実行できる命令を横に並べて記述することにより、あらかじめ、並列論理型言語の命令KL1bに一对一に対応した、RISC命令列の並列パターンを設定しておくようにしたものである。

40 【0007】本発明の請求項3の発明は、請求項1の発明において、並列論理型言語の命令KL1bをベーシック・ブロックと見なし、ベーシック・ブロックの融合の可否を論理型言語の仕様にある並列性とインタフェース・レジスタの依存関係に基づき判定することにより、論理型言語の命令KL1bを並列に記述するようにしたものである。

50 【0008】本発明の請求項4の発明は、請求項3の発明において、ベーシック・ブロックを融合する際に、ベーシック・ブロックの中の、並列に実行される可能性のある並列論理型言語の命令を実行するRISC命令の依存関係を調査することなく、並列なRISC命令列を作

成するようにしたものである。

#### 【0009】

【作用】図1において、例えば、KL1言語などの並列論理型言語1をコンパイラ2によりコンパイルして並列論理型言語のKL1b抽象命令3を生成する。ついで、並列論理型言語の特徴を生かしてKL1b抽象命令3を並列にスケジューリングする。

【0010】一方、KL1b抽象命令3に一つ一つに対応するRISC命令列の並列パターン4を設定しておき、KL1b抽象命令3をベーシック・ブロックとみなして、KL1b抽象命令をRISC命令列に展開する。また、その際、インタフェース・レジスタの依存関係を調べ、依存関係のないKL1b抽象命令3を並列にスケジューリングする。

【0011】本発明においては、上記のようにして、抽象度の高いKL1b抽象命令などの並列論理型言語の命令を、VLIW方式の並列計算機で実行できるRISC命令列に変換しているため、複雑な最適化処理を行うことなく、並列論理型言語をRISC命令レベルで並列化することができ、容易にVLIW方式の命令を作成することができる。

#### 【0012】

【実施例】まず、本発明の前提となるVLIW方式および並列論理型言語の特徴について説明する。

##### 1. 1 VLIW方式について

VLIW方式の命令とは、コンパイラが出力するコード形式の一つであり、VLIWコードを生成するコンパイラが出力する命令がVLIW命令と呼ばれる。

【0013】図2はVLIW命令の形を示す図であり、同図に示すように、VLIW命令の内部には、複数の逐次実行命令が並列に記されている。そして、VLIW方式の並列計算機はVLIW命令を実行の単位として、同図の1, 2, ..., i, ..., mの順序で命令を実行する。これにより、各VLIW命令の内部に並列に記されている複数の逐次命令1ないしnが同時に実行される。

##### 1. 2 並列論理型命令の特徴

a) 並列論理型言語の命令は抽象レベルの高い命令であ \*

```
goukei (A, B, C, D, Z) :- true      | add (A, B, X), add (C, D, Y), add (X, Y, Z).
                                | add (X, Y, Z) :- true      | Z=X+Y.
```

プログラムはゴールが与えられて起動される。上記例において、プログラムの起動をgoukei (1, 2, 3, 4, Z) というゴールで行う。まず、与えられたゴール名と等しいクローズヘッドを持ったクローズが選択され、引数の個数と型があっているかチェックされる。goukeiの場合、与えられたゴールの引数が5個で4つの引数が整数であるので、クローズが選択されるためには、ヘッド部のゴールの引数が与えられた値と等しい数値か、もしくは、構造を持たない変数でなければならない。

【0019】並列論理型言語の言語仕様にある並列性の

＊る。

【0014】並列論理型言語のコンパイラが出力する、ゴール・レベルで並列実行が可能な命令は、抽象度の高い命令である（以降、並列論理型言語のコンパイル結果をKL1b、または、並列論理型言語の命令という）。この命令をVLIWに列挙できる機械語命令に近いレベルの命令（以下、このような命令をRISC命令という）に展開することができる。

【0015】そして、並列論理型言語の個別の命令を、RISC命令列で置き換えるため、一般には、このRISC命令列は入口／出口がそれぞれ一つずつの命令列であるベーシック・ブロックとみなすことが可能であり、このブロック内で並列化スケジューリングを行うことができ、このブロック内のスケジューリングは、静的な情報（実行結果を含まない情報）のみで命令を並列化できる。したがって、並列論理型言語の命令は1対1で並列化スケジューリング済みのRISC命令列と対になる。

b) 並列論理型言語の仕様に並列性が存在する。

【0016】並列論理型言語には、その名前が示すように、言語の仕様として並列性が存在する。本実施例においては、一例として、並列論理型言語KL1の並列性について説明する。KL1はフラットGHC (Flat Guard ed Horn Clauses)に基づいて設計された言語であり、以下、言語の仕様にある並列性を、KL1プログラムの実行の様子と合わせて述べる。

【0017】KL1プログラムは次のシンタックスを持つクローズの集合として表される。

$H: -G_1, \dots, G_m \mid B_1, \dots, B_n. (m \geq 0, n \geq 0)$

ここで、 $H, G_i, B_j$  は各々クローズ・ヘッド、ガード・ゴール、ボディ・ゴールと呼ばれる。オペレータ「 $\mid$ 」はコミットメントバーと呼ばれ、クローズの中でこれに先立つ部分をガード部、これに続く部分をボディ部と呼ぶ。ここで、ガードゴールとしては、組み込み述語しか書けない。また、ボディゴールには組み込み述語とユーザ定義述語の2種類が書ける。

【0018】次に、上記KL1のプログラムがどのようにして実行されるか、下記のプログラムについて簡単に説明する。

特徴の一つはここにあり、①クローズを選択する時に行われる引数の判定は、複数の引数に対して同時に実行できる。この並列性を、以降、ヘッド部のユニフィケーション（ヘッド部における型の判定）と言う。上記例では、プログラムのgoukeiの引数は全て構造を持たない変数であり、与えられたゴールと引数と個数があっているため、goukeiのクローズが選択される。すなわち、上記プログラムにおいて、goukei (A, B, C, D, Z) の引数の型はgoukei (1, 2, 3, 4, Z) と同様、LIST形式などの構造を持つ変数でなく、その個数も5個なので、このクロー

ズが選択される。

【0020】クローズが選択されると、ガード部の述語が実行される。並列論理型言語の言語仕様にある第2の並列性は、②ガード部の複数の述語は同時に実行できるという点にある。しかしながら、上記プログラムにおいては、ガード部は常に成立する述語 (true: 常に真であるので次を実行) が与えられているので、実処理はなく、すぐにボディ部が実行される。

【0021】ボディ部には、ユーザ定義述語の add が3回使われている。並列論理型言語の言語仕様にある第3の並列性は、③ボディ部の述語はすべて同時に実行することが可能であるという点にある。したがって、上記例における add は同時に実行できる (これを子ゴールの生成という)。ただし、3つめの add は1つめと2つめの結果を使用しているため、実際は1つめと2つめの add の結果が決まるのを待って実行される。このように、データの依存関係によって、同時に実行できない場合がある。

【0022】ボディ部の述語 add は上記 goukei と同じように新しいクローズを起動させるゴールとなって、同じゴール名のクローズの選択に移る。例えば、上記例における add(A, B, C) というゴール名と等しいコール名のクローズ・ヘッドを持ったクローズ (次の行の add(X, Y, Z)) が選択される。ここでも、引数の個数と型があっているかの判定が同時に行われ、add のボディ部では、演算の結果を Z に代入する際にユニフィケーション (型の判定) が実行される。

【0023】ここで、④複数のユニフィケーションが存在する場合には、それらは同時に実行でき、これは並列論理型言語の言語仕様にある第4の並列性である (これを、ボディ部のユニフィケーションという)。次に、並列論理型言語 KL1 のコンパイル・コードである KL1b を VLIW 命令列に変換する手順について説明する。

## 2. 1 KL1b の RISC 命令列への展開

KL1b は抽象度の高い命令である。したがって、KL1b 命令を RISC 命令で書き直すと、1つの KL1b 命令は複数の RISC 命令で置き換えられることになる。また、1つ1つの KL1b 命令は当然ながら、入口／出口がそれぞれ一つなので、FORTRAN 等という

【0024】したがって、各々の KL1b 命令単位で対応する RISC 命令列を静的にスケジューリングすることが可能であり、図3に示すように、KL1b 命令と RISC 命令列の対を生成することができる。例えば、KL1b 命令の機能を RISC 命令を用いてコーディングすることにより、KL1b 命令に対応する RISC 命令列を生成することができる。

【0025】上記のように、KL1b 命令と対応づけ、RISC 命令レベルでスケジューリングを施した RISC

C 命令の組をデータとして用意すると、KL1b 命令を並列 RISC 命令列に自動的に変換するシステムを得ることができる。

## 2. 2 同時に実行可能な KL1b 命令の並列記述

並列論理型言語の特徴から同時に実行可能な KL1b 命令を記述する。図4は同時に実行可能な KL1b 命令を並列に記述した一例を示す図であり、同図に示すように、並列論理型言語の特徴から同時に実行可能な KL1b 命令を並列に記述することができる。なお、同時に実行可能な KL1b 命令を並列に記述する際には、データの依存関係をこわさないような注意が必要である。ここで、同時に実行可能だとみなされた KL1b 命令は、FORTRAN 等のトレース・スケジューリングの最適化でいう、ベーシック・ブロックを広げてトレースを大きくとった (2以上のベーシック・ブロックをつなげて1つとする) のと同じ状態となる。

【0026】また、上記スケジューリングを行う際には、インタフェース・レジスタを調査して並列の可能性の可否を判断する。次に示すプログラムは、KL1b 言語で書かれた append プログラムの命令列の一例を示しており、次のプログラムにより並列性の可否の判断について説明する。

【0027】

```
label append/3.
...
put_list 5. (0)
write_car_value 5,4 (1)
write_cdr_variable 5,4 (2)
get_list_value 5,4
put_value 4,3
execute append/3.
```

上記例は、(0) でリスト構造が作られ、(1) と (2) でリストの car 部と cdr 部にデータが書き込まれる命令列である。

【0028】上記例の (1) の write\_car\_value 5,4 はインタフェース・レジスタ5が持つリスト構造の car 部にインタフェース・レジスタ4にあるデータを書き込むことを指示する命令である。また、(2) の write\_cdr\_variable 5,4 はインタフェース・レジスタ5が持つリスト構造の cdr 部とインタフェース・レジスタ4の両方から、新しいデータ領域を指すことを指示する命令である。すなわち、インタフェース・レジスタ5の cdr 部とインタフェース・レジスタ4に新しい変数を指示するアドレスを格納し、新しいデータ領域を指すことを指示する命令である。

【0029】上記命令 (0), (1), (2) においては、インタフェース・レジスタ5を使用する。これをインタフェース・レジスタ5にぶつかりがあるという。ここで、命令 (1) と (2) はレジスタ5を参照するので、インタフェース・レジスタ5にデータを書き込む命

令(0)が終了するのを待って実行されなければならない。また、(1)と(2)の命令でインタフェース・レジスタ4にぶつかりがあるが、命令(1)と命令(2)を並列に並べても、命令の実行に際しては、参照のあとで書き込みが生じ、(2)でインタフェース・レジスタ5に書き込みが生ずるのは(1)の命令でインタフェース・レジスタ4を参照したのちであるので、この二つの命令を同時に実行するのは問題がない。

2. 3 並列に記述されたKL1bのRISC命令での置き換え。

【0030】上記2. 1で述べたように、各々のKL1b命令はRISC命令列に展開することができ、また、上記2. 2で述べたようにKL1b命令はそれらの間に依存関係がなく並列に記述されているから、並列に記述されたKL1b命令をシステムにより自動的にRISC命令列に展開し、RISC命令列の間の依存関係を調べることなく並列のRISC命令列を生成することができる。

【0031】図5は並列に実行可能なKL1b命令1、KL1b命令2、および、KL1b命令3を、それぞれのKL1b命令に対応したRISC命令列に置き換えた場合を示した図であり、同図に示すように並列に記述されたRISC命令列を得ることができる。

2. 4 RISC命令の整理

図5に示す広げられたベーシック・ブロックは、複数のRISC命令の組が並べて置かれた状態になるので、ブロック中でVLIWの並列度に近くなるように、RISC命令列を整理する。

【0032】すなわち、RISC命令の内、横に並べかえることができる命令を横に並びかえ、実質的な並列度を高くする。その結果、図6に示すように、並列論理型言語の命令を、VLIW方式の並列計算機で実行できるようなVLIW命令に変換することができる。次に、下記に示す、ゴールの引数として与えられた2つのリストをつなげる処理を行うKL1で記述されたappendというプログラムを一例として、具体的な実施例について説明する。

```
append([A | X], Y, Z):-true | Z=[ A | Z1], append(X, Y, Z1).
```

```
append([], Y, Z):-true | Z=Y.
```

上記プログラムをコンパイルすると、図7に示すコードのKL1b命令列が得られる。

<実施例>次に、前記2. 1で説明したように、図7に示すKL1b命令を対応するRISC命令列に変換する。なお、図7に示したKL1b命令列に対応したRISC命令を示すと膨大な量になるので、上記appendというプログラムに現れたKL1b命令に対して、対となるRISC命令を示すと図8ないし図11に示すようになる(図3に対応)。

【0033】一方、上記appendというプログラム

のKL1b命令列には図12に示すような並列性がみられる(図4に対応)。そこで、図12に示すKL1b命令をRISC命令で置き換えて、ここでは、VLIWの並列度を3として、RISC命令列を整理すると、図13ないし図16に示すRISC命令列を得ることができる。

【0034】

【発明の効果】以上説明したように、本発明においては、抽象度の高いKL1b抽象命令などの並列論理型言語の命令を、並列論理型言語の特徴を生かして並列にスケジューリングし、スケジューリングされた並列論理型言語の命令を、並列論理型言語の命令に一对一に対応した機械語レベルに近いRISC命令列に展開することにより、VLIW方式の並列計算機で実行できるRISC命令列に変換しているので、複雑な最適化処理を行うことなく、並列論理型言語をRISC命令レベルで並列化することができ、容易にVLIW方式の命令を作成することができる。

【図面の簡単な説明】

【図1】本発明の原理説明図である。

【図2】VLIW命令の形を示す図である。

【図3】KL1b命令とRISC命令列の対を示す図である。

【図4】KL1b命令の逐次実行列を並列に変換した状態を示す図である。

【図5】KL1b命令をRISC命令列に展開した状態を示す図である。

【図6】広げられたベーシック・ブロックの中のRISC命令を示す図である。

【図7】実施例のコンパイル結果を示す図である。

【図8】実施例のKL1b命令とRISC命令列の対を示す図である。

【図9】実施例のKL1b命令とRISC命令列の対を示す図(続き)である。

【図10】実施例のKL1b命令とRISC命令列の対を示す図(続き)である。

【図11】実施例のKL1b命令とRISC命令列の対を示す図(続き)である。

【図12】実施例のKL1b命令を並列に記述した状態を示す図である。

【図13】実施例をRISC命令列に置き換え整理した状態を示す図である。

【図14】実施例をRISC命令列に置き換え整理した状態を示す図(続き)である。

【図15】実施例をRISC命令列に置き換え整理した状態を示す図(続き)である。

【図16】実施例をRISC命令列に置き換え整理した状態を示す図(続き)である。

【符号の説明】

1 KL1言語

- 2 コンパイラ  
3 KL1b命令

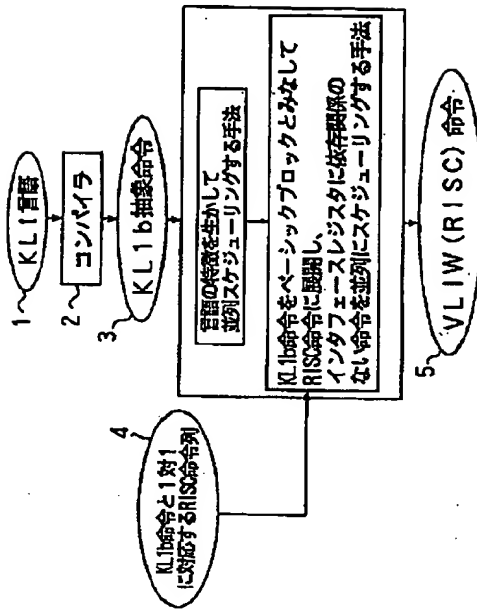
- 4 KL1b命令と1対1に対応するRISC命令列  
5 RISC命令

【図1】

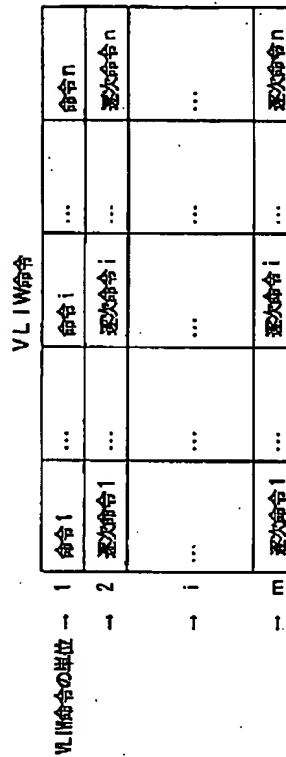
【図2】

【図3】

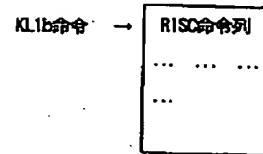
本発明の原理説明図



VLIW命令の形を示す図

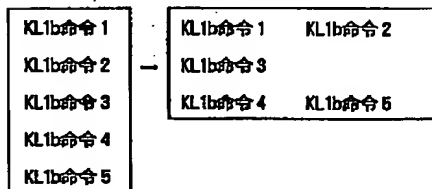


KL1b命令とRISC命令列の対を示す図



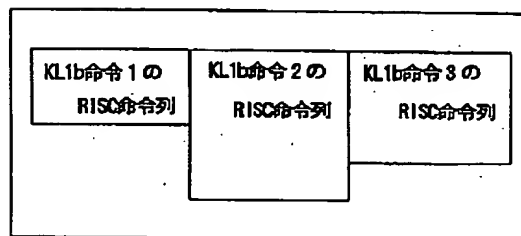
【図4】

KL1b命令の逐次実行例を並列に変換した状態を示す図



【図5】

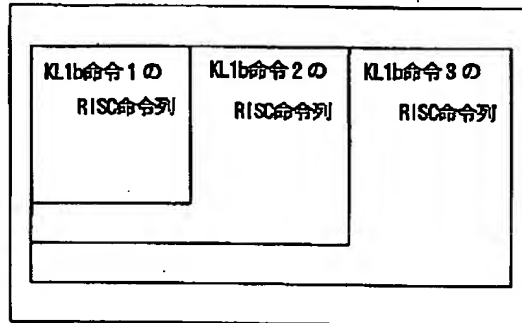
KL1b命令をRISC命令列に展開した状態を示す図





【図6】

広げられたベシックブロックの中のRISC命令を示す図



【図7】

## 実施例のコンパイル結果を示す図

```
label  append/3.  
    try_me_else append/3/1.  
    wait_list 1,append/3/1.  
    read_car 1,4.  
    read_cdr 1,1.  
    put_list 5.  
    write_car_value 5,4.  
    write_cdr_variable 5,4.  
    get_list_value 5,3.  
    put_value 4,3.  
    execute append/3.  
label  append/3/1.  
    try_me_else append/3/2.  
    wait_constant [],1,append/3/2.  
    get_value 2,3.  
    procced.  
label  append/3/2.  
    suspend append/3.
```

【図8】

実施例のKL1b命令とRISC命令列の対を示す図

KL1b 命令	RISC 命令列
try_me_else →	exec acp,(append/3/1,3),IM#1    parbox5cm
wait_list →	exti_u8 1,(Ai,label),IM#1
	refreg 2,1
	slet_m 3,2
	exti_u8 4,3,IM#0
	ceq_u32 5,4,IM#REF    ifundf 3
	br 5,T,PD(3,Ai)    cne_u32 6,4,IM#LIST
	br 6,T,SIU(Ai)    ind_mv 1,3

【図 9】

実施例の K L 1 b 命令と R I S C 命令列の対を示す図 (続き)

wait_constant →	exti_u8 1,(C,Ai,label),IM#2	exti_u16 atom,(C,Ai),IM#1
	ref_reg 2,1	mrgt atom,atom IM#ATOM
	sleq_m 5,2	
	exti_u8 3,5,IM#0	
	ccq_u32 4,3,IM#REF	ifundf 5
	br 4,T,PD(5,Ai)	ind_mv 1,5
	br 6,T,SIU(Ai)	cnc_u32 6,5,atom
	exti_u8 1,(Ai,Aj),IM#1	exti_u8 2,(Ai,Aj),IM#2
	ref_reg 3,1	
	ldb_32 4,3,IM#CAR	
read_car →	ind_mv 2 4	

【図10】

実施例のKL1b命令とRISC命令列の対を示す図(続き)

read_cdr	→	exti_u8 1,(Ai,Aj),IM#1	exti_u8 2,(Ai,Aj),IM#2
		ref_reg 3,1	
		ldb_32 4,3,IM#CDR	
		ind_mv 2 4	
put_list	→	exti_u8 1,(Ai),IM#1	add_u32 2,lctop,IM#0
		mrgt 3,2,IM#LIST	ld_32 lctop,lctop,IM#next
		ind_mv 1,3	
	→	exti_u8 1,(Ai,Aj),IM#2	exti_u8 2,(Ai,Aj),IM#1
put_value		ref_reg 7,2	
		ind_mv 1,7	
	→	exti_u8 1,(Ai,Aj),IM#1	exti_u8 2,(Ai,Aj),IM#2
		ref_reg 4,1	ref_reg 3,2
write_car_value	→	stb_32 3,4,IM#CAR	

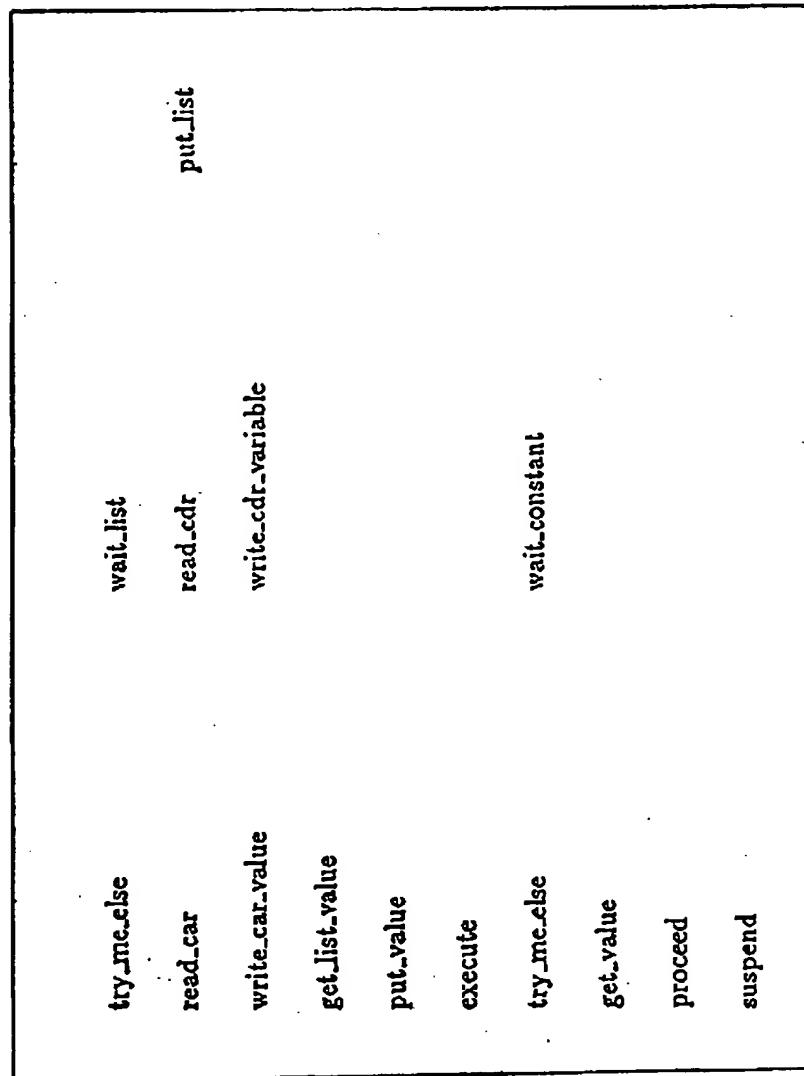
【図11】

実施例のKL1b命令とRISC命令列の対を示す図(続き)

write_cdr_variable →	exti_u8 1,(Ai,Aj),IM#1	exti_u8 2,(Ai,Aj),IM#2	add_u32 3,grtop,IM#0
	refreg 4,1	mrgt 5,3,IM#REF	ld_32 grtop,grtop,IM#next
	std_32 3,5,IM#0		
	stb_32 5,4,IM#CDR	ind_mv 2,5	
get_list_value →	exti_u8 1,(Ai,Aj),IM#1	exti_u8 2,(Ai,Aj),IM#2	
	refreg 3,1	refreg 4,2	
	sct_m 5,4		
	extt_u8 6,5,IM#0		
	cne_u32 7,6,IM#UNDF		
	br 7,T,Getlv(Ai,5,Aj,PCT)		
	mrgt 8,5,IM#REF		
	stb_32 3,8,IM#0		
	ind_mv 2,8		
get_value →	br 0,T,Getlv(Ai,Aj)		

【図12】

実施例のKL1b命令を並列に記述した状態を示す図



【図13】

実施例をRISC命令列に置き換え整理した状態を示す図

extg acp,(append/3/1,3),IM#1	exti_u8 1,(1,append/3/1),IM#1
refreg 2,1	
sldt_m 3,2	
extt_u8 4,3,IM#0	
ceq_u32 5,4,IM#REF	ifundf 3
br 5,T,PD(3,1)	cne_u32 6,4,IM#LIST
br 6,T,STU(1)	ind_mv 1,3
exti_u8 1,(1,4),IM#1	exti_u8 5,(1,1),IM#1
refreg 3,1	exti_u8 9,(5),IM#1
ldb_32 4,3,IM#CAR	add_u32 10,lctop,IM#0
ind_mv 2,4	ld_32 lctop,lctop,IM#next
ldb_32 8,7,IM#CDR	
ind_mv 6,8	ind_mv 9,11
exti_u8 1,(5,4),IM#1	exti_u8 2,(5,4),IM#2
	exti_u8 5,(5,4),IM#1



【図14】

実施例をRISC命令列に置き換え整理した状態を示す図（続き）

refreg 4,1	refreg 3,2	exti_u8 6,(5,4),IM#2
stb_32 3,4,IM#CAR	add_u32 7,grtop,IM#0	
refreg 8,5	mrgt 9,7,IM#REF	ld_32 grtop,grtop,IM#next
std_32 7,9,IM#0		
stb_32 9,8,IM#CDR	ind_mv 6,9	
exti_u8 1,(5,3),IM#1	exti_u8 2,(5,3),IM#2	
refreg 3,1	refreg 4,2	
slct_m 5,4		
exti_u8 6,5,IM#0		
cnc_u32 7,6,IM#UNDF		
br 7,T,Getlv(5,5,3,PCT)		
mrgt 8,5,IM#REF		
stb_32 3,8,IM#0		
ind_mv 2,8		

【図15】

実施例をRISC命令列に置き換え整理した状態を示す図（続き）

exti_u8 1,(4,3),IM#2	exti_u8 2,(4,3),IM#1	
ref_reg 7,2		
ind_mv 1,7		extg_pct,(append/3,3),IM#1
exti_u8 argument_num,(append/3,3),IM#2		
label append/3/1		extia_u16 atom,(0,1),IM#1
extg_acp,(append/3/2,3),IM#1	exti_u8 1,(0,1,append/3/2),IM#2	
ref_reg 2,1	mrgt atom,atom,IM#ATOM	
slct_m 5,2		
extt_u8 3,5,IM#0		
ceq_u32 4,3,IM#REF	ifundf 5	

【図16】

実施列をRISC命令列に置き換え整理した状態を示す図（続き）

